T10/03-176 revision 1

Date: July 25, 2003

To: T10 Committee (SCSI)

From: George Penokie (IBM/Tivoli)

Subject: End-to-End Data Protection

1 Overview

There is an need (real or imagined) for a standardized end-to-end data protection mechanism to be defined. The logical place to such a definition is the SCSI command and architecture standards, as most storage uses SCSI commands to read/write data to and from storage devices. What follows is a proposal that provides a set of SCSI tools that will enable end-to-end data protection. This set of SCSI tools are defined to accomplish this goal:

a) without interfering with existing proprietary methods;
b) with a minimum of options; and
c) by defining minimal changes to CDBs while maintaining backward compatibility.

The set of SCSI tools will consist of the following:

a) Two level data protection on each data block transferred across the interconnect that consists of;
   A) A 4-byte CRC that covers the user data of the data block. The CRC is generated at or before the application client and preserved at the logical unit.
   B) A 4-byte incrementing LBA tag. The incrementing LBA tag is set by the application client during write operation to the value of the least significant 4 bytes of the write command's LBA field on the first data block transferred and incremented by one on each data block transferred until all the blocks for the command have been transferred. The increment LBA tag values for each data block that is read back shall be the same value that was received for that data block.
b) A bit in the non-Read Read CDBs (e.g, PRE-FETCH and REBUILD) to allow a logical unit to return the protection data.
   A) If zero then do not transmit any protection information. The logical unit shall not check the contents of the protection fields.
   B) If one then transmit the protection information. The logical unit may determine if the data block is valid by checking the contents of the protection fields. If the logical unit determines there is a an error it shall generate a check condition. A read to a logical unit that has not been formatted to transmit the data protection fields may fail with a check condition. In the case where the logical unit has not been formatted and does not check the bit the contents of the protection information is unpredictable and as a result should cause an error at the application client.
c) A two bit field in the READ commands (excluding the READ (6) command) that would control the reading and checking of protection data.
   A) If 00b then do not transmit any protection information. ~~The logical unit shall may not check the contents of the protection fields~~ If the logical unit has been formatted with protection information the logical unit may determine if the data block is valid by checking the contents of the protection fields. If the logical unit determines there is a an error it shall generate a check condition.
   B) If 01b then transmit the protection information. The logical unit may determine if the data block is valid by checking the contents of the protection fields. If the logical unit determines there is a an error it shall generate a check condition. A read to a logical unit that has not been formatted to transmit the data protection fields may fail with a check condition. In the case where the logical unit has not been formatted and does not check the bit the contents of the protection information is unpredictable and as a result should cause an error at the application client.
   C) If 11b then transmit the protection information. The logical unit shall not check the contents of the protection fields. A read to a logical unit that has not been formatted to transmit the data protection fields may fail with a check condition. In the case where the logical unit has not been formatted and does not check the bit the contents of the protection information is unpredictable and as a result should cause an error at the application client.
d) A bit in Write CDBs to allow the protection information to be written with no checks.

A) If 00h then preserve the contents of the protection fields (e.g., write to media, store in non-volatile memory, recalculate on read back). The logical unit shall determine if the data block is valid by checking the contents of the protection fields. If the logical unit determines there is a an error it shall generate a check condition. If the logical unit has not been formatted to accept protection information it shall generate a check condition.
B) If 01h then preserve the contents of the protection fields (e.g., write to media, store in non-volatile memory, recalculate on read back). The logical unit shall not check the contents of the protection fields. If the logical unit has not been formatted to accept protection information it shall generate a check condition.
C) If 10h then the contents of the protection fields shall not be preserved. The logical unit shall determine if the data block is valid by checking the contents of the protection fields. If the logical unit determines there is a an error it shall generate a check condition. In the case where the logical unit has not been formatted with protection fields and does not check the CDB protection field the logical units response to the command is unpredictable.

e) A bit in the Format CDB to cause 8 bytes to be added to the block size of the logical unit being formatted.
   A) If zero then format the medium to the block length defined in the mode parameter block descriptor of the Mode parameter header.
   B) If one then format the medium to the block length defined in the mode parameter block descriptor of the Mode parameter header plus 8 (e.g., if block length = 512 the formatted block length is 520). The block length shall be a multiple of four. If the block length is not a multiple of four the logical unit shall generate a check condition.
f) All commands that request block length information (e.g., Read Capacity, Mode Sense) shall return the block size of the data excluding the eight bytes of protection information (e.g., a 520 byte data block on a device formatted with protection information returns 512 in the block length field).
g) A two bit field in the Standard Inquiry Data to indicate support of protection data.
   A) If 00b then no protection is supported.
   B) If 10b then protection is supported but not enabled
   C) If 11b then protection is supported and enabled.
h) A bit in a mode page that forces the logical unit to write to media the contents of the crc field.
   A) If zero then the contents of the crc field shall be preserved and may be written to media.
   B) If one then the contents of the crc field shall be written to media.

SBC-2 additions

4.x Data protection model

4.0.1 Data protection overview

This data protection model provides for the protection the data while it is being transferred between a sender and a receiver. The protection data is generated at the application layer and may be checked by any object along the I_T_L nexus. Once received, the protection information shall be retained (e.g., write to media, store in non-volatile memory, recalculate on read back) by the device server until overwritten (e.g., power loss and reset events have no effect on the retention of protection information).

4.0.2 Protected data

The protection data consists of two fields appended to each block of data that contains:

a) a cyclic redundancy check (CRC); and
b) a logical block address tag.

See figure 1 for the placement of the CRC and LOGICAL ADDRESS TAG fields.

.

Table 1 — Protected data block format

Byte\Bit	7	6	5	4	3	2	1	0
0	DATA BLOCK							
n								
n + 1	(MSB)	CRC						
n + 4								(LSB)
n + 5	(MSB)	LOGICAL ADDRESS TAG						
n + 8								(LSB)

The data block shall contain user data.

The crc field contains the CRC (see 4.0.3) of the contents of the DATA BLOCK field.

The LOGICAL ADDRESS TAG field is set to the least significant four bytes of the logical block address to which the data block is to be associated with. The first data block transmitted shall contain the least significant four bytes of the logical block address contained in the LOGICAL BLOCK ADDRESS field of the command associated with the data being transferred. Each subsequent data block's LOGICAL ADDRESS TAG field shall contain the logical address tag of the previous data block plus one.

4.0.3 CRC protection

If data protection is enabled, the application client shall append a CRC to each block of data to be transmitted. The CRC shall be generated from the contents of the DATA BLOCK field.

Annex C contains information on CRC generation/checker implementation.

Table 2 defines the CRC polynomials.

Table 2 — CRC polynomials

Function	Definition
F(x)	A polynomial of degree k-1 that is used to represent the k bits of the data block covered by the CRC. For the purposes of the CRC, the coefficient of the highest order term shall be the MSB of the DATA BLOCK field.
L(x)	A degree 31 polynomial with all of the coefficients set to one: $L(x) = x^{31} + x^{30} + x^{29} + \ldots + x^2 + x^1 + 1$ (i.e., L(x) = FFFFFFFFh)
G(x)	The standard generator polynomial: $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ (i.e., G(x) = 1_04C11DB7h)
R(x)	The remainder polynomial, which is of degree less than 32.
P(x)	The remainder polynomial on the receive checking side, which is of degree less than 32.
Q(x)	The greatest multiple of G(x) in $(x^{32} \times F(x)) + (x^k \times L(x))$
Q'(x)	$x^{32} \times Q(x)$
M(x)	The sequence that is transmitted.
M'(x)	The sequence that is received.
C(x)	A unique polynomial remainder produced by the receiver upon reception of an error free sequence. This polynomial has the value: $C(x) = x^{32} \times \frac{L(x)}{G(x)}$ $C(x) = x^{31} + x^{30} + x^{26} + x^{25} + x^{24} + x^{18} + x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x + 1$ (i.e., C(x) = C704DD7Bh)

4.0.4 CRC generation

The equations that are used to generate the CRC from F(x) are as follows. All arithmetic is modulo 2.

CRC value in data block = L(x) + R(x) = one's complement of R(x)

NOTE 1 - Adding L(x) (all ones) to R(x) produces the one's complement of R(x); this equation is specifying that the R(x) is inverted before it is transmitted.

The CRC is calculated by the following equation:

$$\frac{(x^{32} \times F(x)) + (x^K \times L(x))}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

The following equation specifies that the CRC is appended to the end of F(x):

$M(x) = x^{32} \times F(x) + CRC$

The bit order of F(x) presented to the CRC function is MSB to LSB four bytes at a time until the contents of the DATA BLOCK field are all processed. This order is shown in figure 1

.

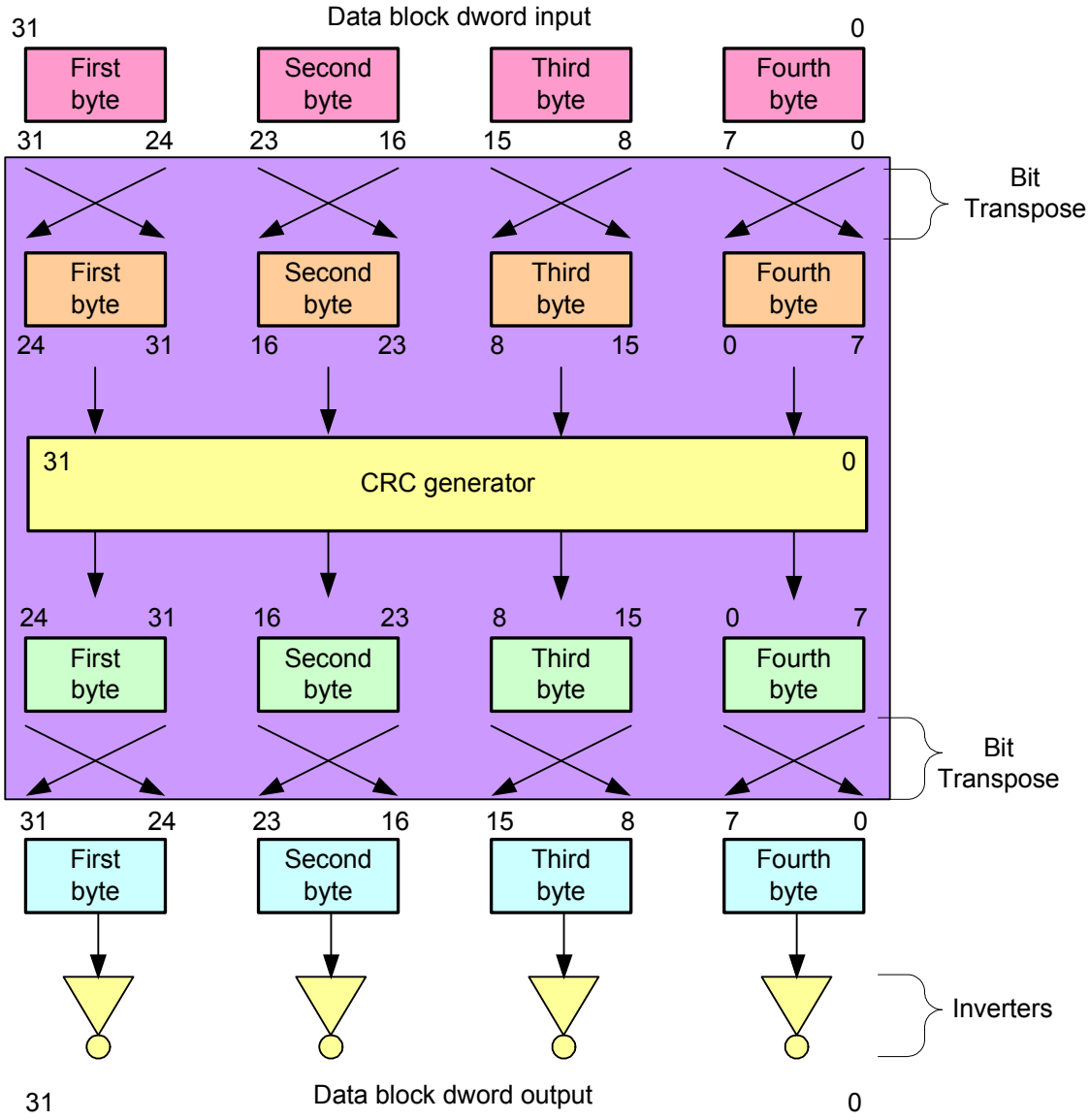

Figure 1 — CRC generator bit order

4.0.5 CRC checking

The received sequence M'(x) may differ from the transmitted sequence M(x) if there are transmission errors. The process of checking the sequence for validity involves dividing the received sequence by G(x) and testing the remainder. Direct division, however, does not yield a unique remainder because of the possibility of leading zeros. Thus a term L(x) is prepended to M'(x) before it is divided. Mathematically, the received checking is shown by the following equation:

$$x^{32} \times \frac{M'(x) + (x^{\kappa} \times L(x))}{G(x)} = Q'(x) + \frac{P(x)}{G(x)}$$

In the absence of errors, the unique remainder is the remainder of the division as shown by the following equation:

$$\frac{P(x)}{G(x)} = x^{32} \times \frac{L(x)}{G(x)} = C(x)$$

The bit order of F(x) presented to the CRC checking function is the same order as the CRC generation bit order (see figure 1).

4.0.6 Application of protected data

Before an application client transmits or receives protected data it shall:

1) Determine if a logical unit supports protected data using the INQUIRY command (see x.x.x);
2) If protected data is supported then determine if the logical unit is formatted to accept protected information using the INQUIRY command (see x.x.x);
3) If the logical unit supports protected information and is not formatted to accept protected information then format the logical unit with protected information usage enabled;
4) If the logical unit supports protected information and is formatted to accept protected information then the application client may use read commands that support protected information and shall use write commands that support protected information.

4.0.7 Protected data commands

The enabling of protection information enables fields in some commands that instruct the device server on the handling of the protection information. The detailed description of each commands protection information fields are defined in the individual command descriptions.

The commands that are affected when protection information is enabled are:

a) EXTENDED COPY (See SPC-3);
b) FORMAT UNIT; (Editing Note: Byte 1, Bit 7 - If the Initialization Pattern is used then the target needs to generate valid protection information)
c) INQUIRY (See SPC-3);
d) PRE-FETCH (10)/(16); (Editing Note: Byte 1, Bit 7)
e) READ (6)/(10)/(12)/(16); (Editing Note: Byte 1, Bits 7 and 6 except for READ (6))
f) READ LONG; (Editing Note: Byte 1, Bits 7 and 6)
g) REASSIGN BLOCKS; (Editing Note: Need comment in command description that the reassigned block needs valid protection information written with it.)
h) REBUILD (16)/(32); (Editing Note: Byte 1, Bit 7on (16) and Byte 10, Bit 7 on (32) - The logical unit needs to generate protection information on the rebuilt data)
i) REGENERATE (16)/(32); (Editing Note: Byte 1, Bit 7 on (16) and Byte 10, Bit 7 on (32))
j) SYNCHRONIZE CACHE (10)/(16); (Editing Note: Need comment in command description that the blocks that are synchronized need valid protection information written with it.)
k) VERIFY (10)/(12)/(16);(Editing Note: Need comment in command description that when blocks are verified the logical unit may use protection information as part of the verification. If it fails because of the protection information then a new ASCQ needs to be returned.)
l) WRITE (6)/(10)/(12)/(16); (Editing Note: Byte 1, Bit 7 and 6 except for WRITE (6))
m) WRITE AND VERIFY (10)/(12)/(16); (Editing Note: Byte 1, Bit 7and 6 , also need comment in command description that when blocks are verified the logical unit may use protection information as part of the verification. If it fails because of the protection information then a new ASCQ needs to be returned.
n) WRITE LONG; (Editing Note: Need comment in command description that the data to be written needs to include protection information.)
o) WRITE SAME (10)/(12); (Editing Note: Need comment in command description that as the blocks of data are written the logical unit needs to generate and write valid protection information.)
p) XDREAD (10)/(32); (Editing Note: Byte 1, Bit 7 and 6 on (16) and Byte 10, Bit 7 and 6 on (32))
q) XDWRITE (10)/(32); (Editing Note: Byte 1, Bit 7 and 6 on (16) and Byte 10, Bit 7 and 6 on (32))
r) XDWRITE EXTENDED (16)/(32)/(64);(Editing Note: Byte 1, Bit 7 and 6 on (16) and Byte 10, Bit 7 and 6 on (32) and (64))
s) XDWRITEREAD (10)/(32); and (Editing Note: For the Write control Byte 1, Bit 7 and 6 on (16) and Byte 10, Bit 7 and 6 on (32) and (64). For the Read control Byte 1, Bit 6 on (16) and Byte 10, Bit 6 on (32) and (64))
t) XPWRITE (10)/(32). (Editing Note: Byte 1, Bit 7 and 6 on (16) and Byte 10, Bit 7 and 6 on (32))

If a WRITE (6) command is received after protection information is enabled the device server shall return CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and an additional sense code of INVALID COMMAND OPERATION CODE.

A READ (6) command may be sent to a logical unit that has protection information enabled but there is no option available to transmit the protection information.

Commands that result in the return of the length in bytes of each logical block (e.g., MODE SENSE, READ CAPACITY) shall return the length of the user data and shall not include the length of the protection information (e.g., if the user data plus the protection information is equal to 520 byte the length in bytes returned is 512 bytes).